Journal of Theoretical and Applied Information Technology
20th August 2014. Vol. 66 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

# NETWORK INTRUSION DETECTION SYSTEM: AN IMPROVED ARCHITECTURE TO REDUCE FALSE POSITIVE RATE

[1] P.BRINDHA, [2] Dr.A.SENTHILKUMAR

[1] Assistant Professor, Department of Electronics and Communication Engineering, Velalar College of Engineering and Technology, Erode-638012, Tamil Nadu, India
[2] Professor and Head, Department of Electrical and Electronics Engineering, Dr.Mahalingam College of Engineering and Technology, Pollachi-642001, Tamil Nadu, India
E-mail: [1]brindhavlsi@gmail.com, [2] ask_rect@yahoo.com

**ABSTRACT**

Counting Bloom Filter (CBFs) is widely employed in networking applications. They support membership queries with minimal error and surmount the drawback of Bloom filters. However, they engross large amount of memory. A new Sidon sequence based CBF is introduced to improve the competency of the CBFs. Unlike CBF, the hashed Variable increment is queried for its present. This method achieves 24% of the improvement in false positive rate and a lower inundate probability bound than CBF. The proposed work is described in Verilog and simulated using Xilinx 12.1. The functional block is implemented as hardware using Virtex 4 (XC4VSX25) Field Programmable Gate Array (FPGA). It is observed that the hardware complexity and memory overhead increases, which could be a trade-off for improving FPR in order to increase the Network Security.

**Keywords:** *Intrusion Detection, Bloom Filter, Sidon sequence, Counting Bloom filter, False Positive Rate.*

## 1. INTRODUCTION

Information is always deliberated to be the most valuable liability of any organization or as an individual and hence, it should be secured. However, the rapid proliferation of the internet and web applications has increased the threat of information security breaches. Traditional mechanisms are often not adequate to protect the data against new attacks. Such attacks are detected by an Intrusion Detection System (IDS). IDS chosen for such action should be flexible, memory efficient and maintaining database is not so much easy in practice. To enhance the detection and to improve the false positive rate of Counting Bloom Filter(CBF), a new Sidon sequence based CBF is presented in this paper.

### 1.1 Bloom Filter

Burton H. Bloom introduced a new hash coding method. This method is suggested for application in which the great majority of messages to be tested will not belong to the large set. First, the average time required for classifying the element as a non-member of large set is high. Second, the probability of error should be minimized (i.e.) the false identification of the member to be in the set will create a small error. Third, computation time and space should be efficient to meet the practical applications [1],[2],[3].

Consider a set S = $\{a_1,a_2,a_3\ldots\ldots a_n\}$ of 'n' elements and a set H = $\{h_1,h_2,h_3\ldots\ldots h_k\}$ of 'k' independent and uniformly distributed hash function. The individual hash, say $h_1$ will range from {0,1,2,……m-1}. Consider an example: If S = {AA,AB,BA,BB……ZZ} where n=100 and assume k = 3, then H = $\{h_1,h_2,h_3\}$. Using 'k' hash function of H makes each element of S map to the vector whose size is 'm'. An array is constructed whose length is 'm'. The initial value in the array is maintained at zero as depicted in Figure 1. As the hash function starts, depending on the hash output the corresponding locations are set to one as in Fig.2. On the other hand, the query is done to check whether the element belongs to set [4],[5].

The problem with the standard BF is, the same location is accessed for multiple times based on the hash output as shown in Figure 2.The item deletion always required to make it flexible for many applications. There are three operations awaited by an IDS user.

- Signature insertion: to maintain a database of the existing virus signatures and eventually include new items whenever it is required.

Journal of Theoretical and Applied Information Technology
20th August 2014. Vol. 66 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

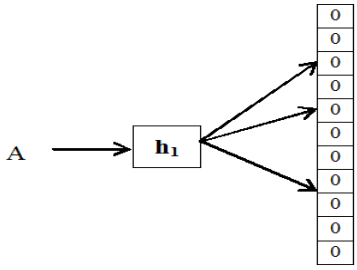



*Figure 1: Empty Bloom Filter At The Initial Condition*

- Signature deletion: this is to revoke the items when it is found to be harmless.
- Signature matching: to give a diligence to the user.

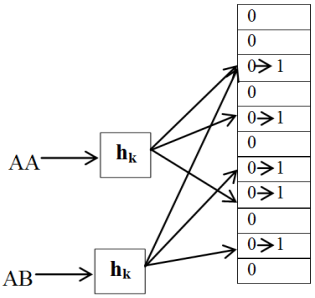



*Figure 2: Bloom Filter After Insertion Of 2 Elements*

A flawless BF should perform all the three about said operations. But unfortunately BF can perform only first and last operation. As shown in Figure 3, if multiple items are hashed to the same index, while deletion of one item may cause changes in the hash index of other. This shortcoming of BF makes it obsolete in many applications.

The hash locations $h_i(a)$ is checked to ensure the presence of one in all the locations that are hashed. If the presence is confirmed, then $a \in S$ and if not, then $a \notin S$. The querying is shown for a element B in Figure 4. Furthermore, if all the hash values mapped to the vector is '1' but, the element says 'AA' is not a member of the set, then this is referred as False Positive. Hereafter the main focus is on False Positive Rate (FPR), which is to be reduced to enhance the performance of the IDS.

### 1.2. False Positive Rate (FPR) of BF

The False Positive Rate (FPR) can be expressed as

$$fp_r = (1 - P_0)^k \quad (1)$$

Where, $fp_r$ is the False Positive Rate (FPR) and $P_0$ is the probability that all the bits are set to '0'.

Then

$$P_0 = (1 - \frac{1}{m})^{kn} \quad (2)$$

Where, $\frac{1}{m}$ is the random bit of the vector is set to '1' by hash function $h_i$. Then $P_0$ can be approximated to

$$P_0 \cong e^{\frac{-kn}{m}} \quad (3)$$

An element is said to be false positive, only if $\forall[h_i(a)] = 1$, then it can be expressed by

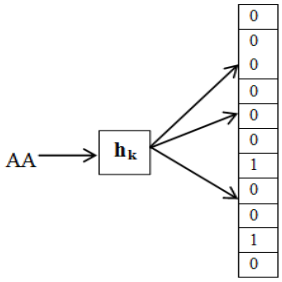



*Figure 3: Bloom Filter After Deleting 'AA'*

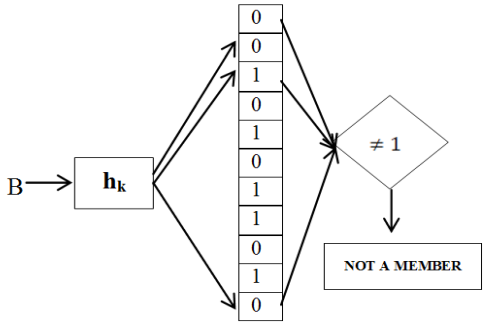



*Figure 4: Querying A Signature 'B'*

$$fp_r = \left[1 - (1 - \frac{1}{m})^{kn}\right]^k \quad (4)$$

$$\cong (1 - e^{\frac{-kn}{m}})^k$$

$$= e^{k \ln(1 - e^{\frac{-kn}{m}})} \quad (5)$$

Journal of Theoretical and Applied Information Technology
20th August 2014. Vol. 66 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$$\min(fp_r) = (1/2)^{k \min} \quad (6)$$

$$= 0.619 \frac{m}{n} \quad (7)$$

$$k = \left\lfloor \ln 2 \frac{m}{n} \right\rfloor \quad (8)$$

The False Positive can be reduced by increasing the value of m/n. But increasing the value of 'm' will increase the memory space and reducing the term 'n' will scale down the set S. Both these actions abet to many detriments in intrusion detection applications.

## 2. COUNTING BLOOM FILTER (CBF)

Counting Bloom Filter emerged to overcome the pitfalls of BF. CBF have been contrived for multi-set representation that may be dynamic due to increment/decrement. This aspect makes it as a natural virtue in many networking solicitations [6], [7].

### 2.1. Functionality of CBF

The BF uses an imperfect representation of the large set to be searched. CBF has a counter array which is indexed by the hashing the input signature. In case of multiple entries the counter gets incremented to keep track of it. The two important facets of CBF are:

- Deletion of a signature, which is not facilitated in BF.
- To cognizant on the multiple access.

Ideally, CBF has a discrete entry which would exist for each single element of the set as in Figure 5. In this case, the CBF would be capable of precisely representing any set [8].

Before measuring the false negative items, let us recall the four rubrics to delete an item 'AA' from a pertinent CBF of a set S.

- If a membership query for an item $a \in S$ acknowledges a correct answer, the CBF performs the item deletion operation by decrementing corresponding counters by one.
- If a membership query for an item $a \in S$ acknowledges a false negative, the CBF rejects the item deletion operation. It shows that the CBF does not emulate the set S correctly.

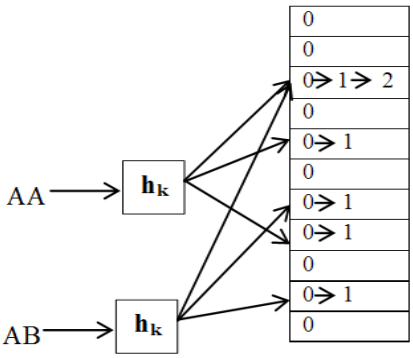



*Figure 5: CBF After Inserting 2 Elements*

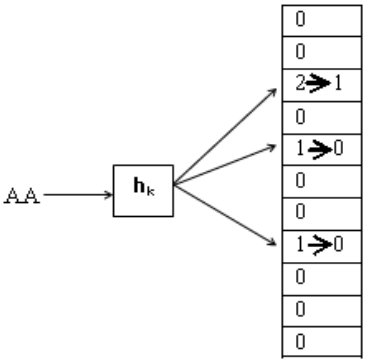



*Figure 6: CBF After Deleting 'AA'*

- If a membership query for a $\notin$ S acknowledge a correct answer, the CBF ignores the item deletion operation.
- If a membership query for a $\notin$ S response a false positive judgment, the CBF still performs the item deletion operation.

### 2.2. False Positive Rate of CBF

According to the CBF basic theory, the false positive probability of a CBF should decrease, if it performs an item deletion operation. Thus, the second point indicates it may increase the false positive probability of a CBF as an item is not deleted, although it should be deleted. In summary, the fourth point implies not only it produces false negative items directly, but also it may increase the probability of false positive judgments obliquely.

In Figure 6, a signature 'AA' is deleted from the CBF (i.e.) the count is deduced from '1' to facilitate the item deletion. When the element 'AB' is queried, the CBF provides an alert that it is a member. Even though the CBF has enhanced feature compared to BF there is a little hindrance in CBF too.

Journal of Theoretical and Applied Information Technology
20th August 2014. Vol. 66 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The foremost problem in CBF is signature identification. As the signature database is maintained as counter value, there is no definite way to differentiate the counting value in terms of signature. CBF suffers from counter overflow with a probability that depends on the size of the counter. To overcome all these flaws in CBF, a new design based on Sidon sequence came into the scenario [9].

## 3. SIDON SEQUENCE

In this section we introduce the Sidon Sequence which is commonly known as *Bh*-set or *Bh* sequence [10]. We start with the basic Scenarios of *Bh* sequences. *B*2 sequences are also called Sidon sequences.

***Rule 1): Let (X; +) be an Abelian group.***

Let $G = \{\mathcal{U}_1, \mathcal{U}_2 \ldots \mathcal{U}_n\}$ be a sequence of elements of *X*. Then *G* is a *Bh sequence* over *X* if all the sums $\mathcal{U}_{i1} + \mathcal{U}_{i2} + \cdots\cdots + \mathcal{U}_{ih}$ with $1 \leq i_1 \leq i_2 \ldots \leq i_h \leq n$ are distinct.

***Example 1:*** Let $G = \mathbb{Z}$ and $G = \{\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4\} = \{1,4,11,13\} \subseteq G$. We can see that all the 10 sums of 2 elements of *G* and 20 sums of 3 elements are distinct. Therefore, *G* is a *B2* and *B3* sequence. However, the two sums of 4 elements are repeated as shown in Table 1, hence *G* is not a *B4* sequence.

### 3.1. Functionality Of Bh Sequence

The *Bh* sequence can be now used to improve the performance of CBF. Basic CBF will increment the counter by 1, which is constant throughout the entries. While in *Bh* sequence based CBF there will be two counters [10].

- First counter will have a constant increment of one, it counts the number of elements hashed into the CBF.
- Second counter will have an inconstant incremental of weighted sum of the elements as shown in Table.1.

There are three basic operations performed by each of these counters.

- Insertion or update phase
- Deletion or removal phase
- Query or test phase

Figure 7.(a,b,c) indicates the basic operation with an example.

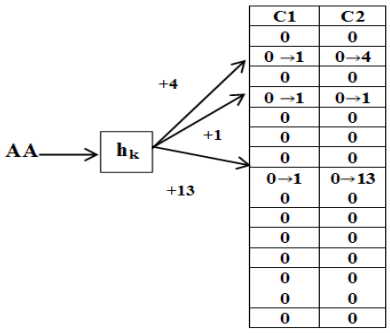



*Figure 7: A) $B_h$ Element Insertion*

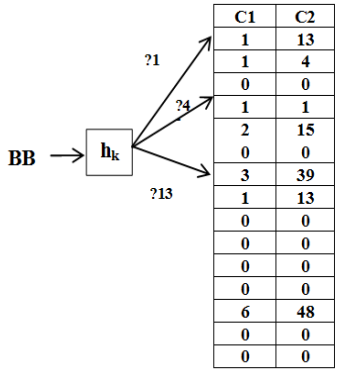



*Figure 7: B) $B_h$ Element Query.*

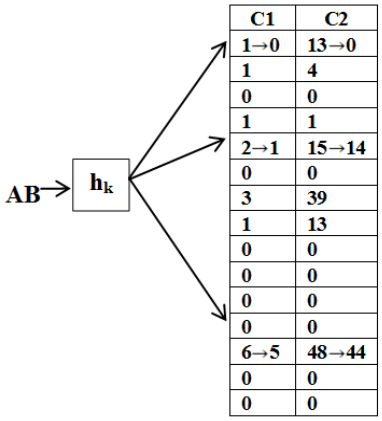



*Figure 7: C) $B_h$ Element Deletion.*

**3.1.1 Element insertion***:* Here in this example an element 'AA' is inserted whose G = {1,4,11,13} and k = 3. In this, $\{h_1(AA),h_2(AA),h_3(AA)\} = \{1,4,8\}$ and $\{g_1(AA),g_2(AA),g_3(AA)\} = \{2,1,4\}$. The element 'AA' is hashed to 1,4,8 locations in the array using $h_1$, $h_2$ and $h_3$. It increment the counter C1 by one and it increments the second counter by hashing $\{\mathcal{U}_{g1(AA)}, \mathcal{U}_{g2(AA)}, \mathcal{U}_{g3(AA)}\} = \{\mathcal{U}_2, \mathcal{U}_1, \mathcal{U}_4\} = \{4,1,13\}$ respectively.

Journal of Theoretical and Applied Information Technology
20th August 2014. Vol. 66 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

**3.1.2 Element querying:** To query whether an element 'AB' is present in the set S, both the counters are checked for its presence.

- If C1($i$)=0 then the constant increment counter determines that 'AB' ∉ S.
- The C2($i$) is checked for exact sum of the a value based on $G$.

**3.1.3. Element deletion:** The deletion of an element says 'AB' is performed similar to element insertion. First counter of $h_i(a)$ is decremented by one and second counter is decremented by $\mathcal{U}_{gi}(a)$. For example, as indicated in Figure7. the element 'AB' is removed by decrementing the first counter C1 using $h_1$(AB),$h_2$(AB) and $h_3$(AB) by 1 and their second counter C2 by 13,1 and 4 respectively.

### 3.2. False Positive Rate (FPR)

False Positive Rate (FPR) of CBF is given by assuming $n$ as the number of elements, $m$ is the number of counters and $k$ is the hash function.

$$FPR = \left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k}$$

$$= \left(1-\left(1-\frac{1}{m}\right)^{\ln 2m}\right)^{k} \quad (9)$$

When m→ ∞

$$FPR = \left(\frac{1}{2}\right)^{k} \quad (10)$$

To avoid overflow, the precise counter should be used to avoid errors. For four bit counter, probability to avoid overflow is

$$\Pr(\max(ci) \geq 16) \leq m\left(\frac{enk}{16m}\right)^{16} = 1.37 \times 10^{-15} \cdot m$$

False Positive Rate (FPR) of *Bh-based Bloom filter* given by Ori Rottenstreich et al., [10] is:

$$FPR = \left(1-\sum_{j=0}^{h}\binom{nk}{j}\left(\frac{l-1}{lm}\right)^{j}\left(1-\frac{1}{m}\right)^{nk-j}\right)^{k} \quad (11)$$

The probability at X=j which indicates the number of elements hashed into the indices.

$$Pr(X=j) = \binom{nk}{j}\left(\frac{1}{m}\right)^{j}\left(1-\frac{1}{m}\right)^{nk-j} \quad (12)$$

The probability that an element, x $\notin S$ cannot deduce from each of the $k$ hash entries that x $\notin S$ is,

$$FPR = \left(1-\sum_{j=0}^{h} Pr(X=j)\left(1-\frac{1}{l}\right)^{l}\right)^{k} \quad (13)$$

By substituting appropriate values for in the equation (10) and (12) it can be seen that the false positive rate reduces as there is an increase in the value of $h$, $l$.

FPR is 1 when m→∞. As k varies, the FPR varies as pointed in the Figure 8, there is a significant variation in the plot, as $k$ increases the FPR of the CBF decreases. The theoretical values for different scenarios are displayed in Table 2. Based on the equation (12), for any value of $k$, $m$, $n$ the positive rate always decreases when $h$ and $l$ are increased. The plot shown in Figure 9, directs the disparity of FPR for different values of $h$ (at $l$=15, $l$=5).

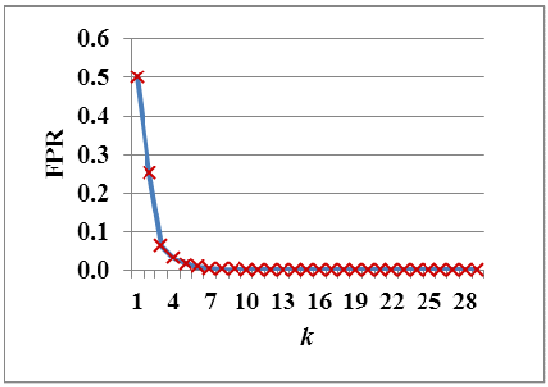



*Figure 8: The Probability Function Of CBF Assuming M Tends To Infinity*

*Table 2: Bh Sequence Variations By Holding N=100, M=16 As Constraints*

| *h* | *l* | *k* | *FPR* |
|---|---|---|---|
| 1 | 15 | 2 | 0.99999 |
| 8 | 5 | 2 | 0.95010 |
| 15 | 4 | 5 | 0.99997 |
| 4 | 3 | 1 | 0.75177 |
| 11 | 12 | 1 | 0.76459 |

From Table 2, the false positive rate is minimum for $h$=4, $l$=3 and $k$=1.

## 4. IMPLEMENTATION RESULTS

In this section, the discussion is on implementation of CBF and $B_h$ architecture.

Journal of Theoretical and Applied Information Technology
20th August 2014. Vol. 66 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Upon discussion several issues are considered like bits/cycle, throughput and number of LUTs required for different hashing complexities. The proposed work is described in Verilog and simulated using Xilinx 12.1. The functional block is implemented as hardware using Virtex 4 (XC4VSX25) Field Programmable Gate Array (FPGA).

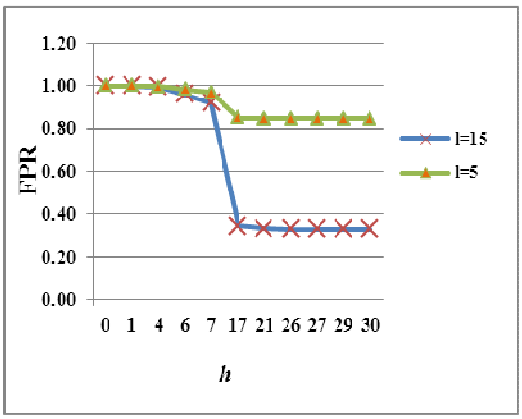



*Figure 9: The Probability Function Of Bh Sequence For N=100, K=2, M=16*

Figure 10, Presents the implementation of CBF in which the flow ID of the packet is hashed into one of the CBF array entries. The corresponding Linear Shift register based counter value C1 in the Look-up array are considered [11]. If it equals zero, CBF determines that $a \notin S$. If it equals one, then CBF continues to check the next hash entry.

Figure 11, Illustrates the implementation of $B_h$ sequence architecture. Components that are present in CBF are also present here and in addition to that few new components are incorporated to enable the use of two hash functions instead of one. The first hash function points to an entry in the $B_h$ architecture array with the pair of counters ($C1$, $C2$) and the second hash function points to an increment from the set $G$ denoted by $\mathcal{U}$.

To powerfully determine whether the weighted sum of C2 whose value is comprised of $\mathcal{U}$ using C1 addends we suggested table based on G. Based on the entry in C1 and the weighted sum of addends in C2, the presence of packet is determined.

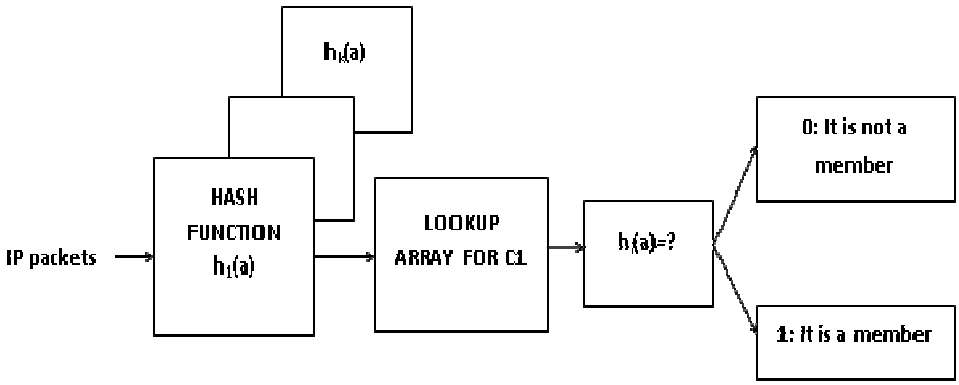



*Figure 10: Functional Module For Counting Bloom Filter*

Journal of Theoretical and Applied Information Technology
20th August 2014. Vol. 66 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

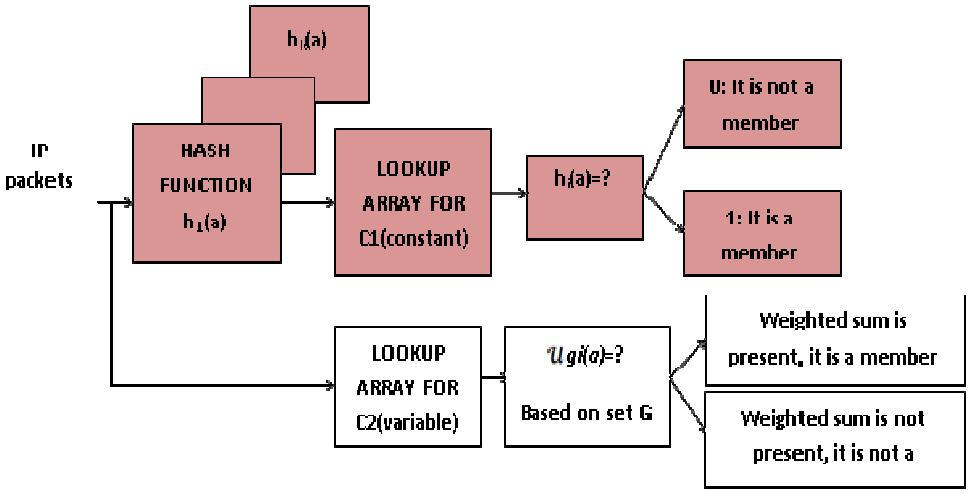



*Figure 11: Enhanced Counting Bloom Filter with $B_h$ Sequence*

The FPR increases as increases in the number of entries. As the hash component *(1,2,3…. k)* is increasing the FPR decreases with little expense in speed.

Many real time domains that depends on the advantages of Bloom filters are Authentication, Firewalling, Anomaly detection, Trace backing, Node replication detection, Anonymous routing, privacy-preserving, String matching, DoS and DDoS addresses, Email protection, and misbehavior detection. Upon this CBF found its area in Firewalling, String matching, Email protection, SYN flooding addressing etc.,

A hardware module for $B_h$ is presented in Figure. As narrated in previous sections two hash functions are used. Blocks similar to CBF are shown in pink color, and the second hash in shown in white color. The weighted sum is monitored to make decision on the incoming query. The experimental result shows the variation in throughput, number of entries and LUTs required.

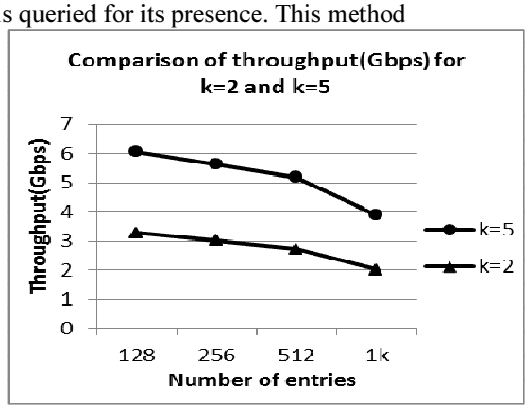
As there is aggrandize in number of entries the throughput decreases rapidly. For 128 entries the throughput in Gpbs achieved is 3.26 and 2.81 for **k=2** and **k=5** respectively. Variation is duly by the rise in **k** (i.e. hash function). Another important factor is LUT requirement.

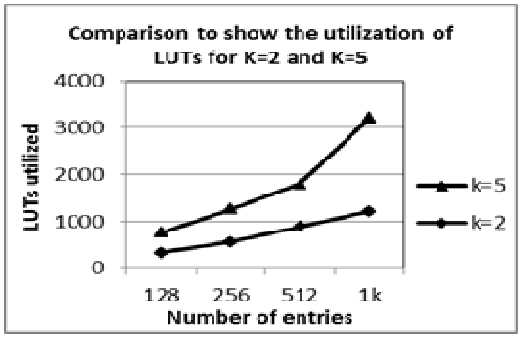
Extend in number of entries gives rise to the new upgraded counter to uphold the new entries. Due to which the LUTs also increases from 331 to 428. As the number of entries is doubled the throughput is reduced to 3.02Gbps, which is around 7.3% less. Similar degraded throughputs are observed for different entries as shown in Table 3.

Next looking upon the LUTs required, there is a huge drift from 331 to 1219 and from 428 to 1990 for k=2 and k=5 respectively.

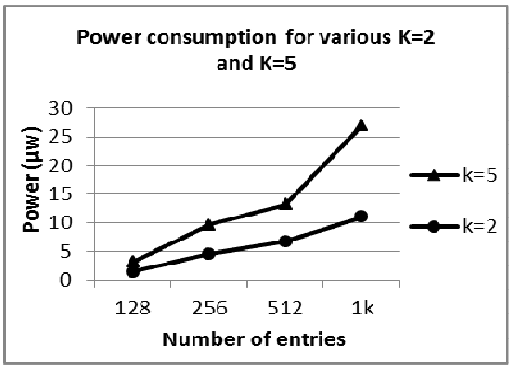
## 5. CONCLUSION

In this paper, we presented a new Sidon sequence based CBF to improve the competency of the CBFs. Unlike CBF, the hashed Variable increment is queried for its presence. This method



ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195




achieves 24% of the improvement in false positive rate and a lower inundate probability bound than CBF. We showed that it can be resourcefully implemented in hardware. Although the FPR improves with added complexity, memory overhead has to be considered while the array size increases. The results show that the throughput is increased from 2.01 to 3.26Gbps as the number of entries is reduced from 1000 to 128. Similarly the area complexity (ie)., the number of LUTs utilization gets reduced as the hash is kept low. Always the networking applications require complex coding techniques and increased h states to secure the data. Sidon sequences efficiently compress the h states so it can be successfully used for network applications with low complexity. In future this work can be extended to optimize memory requirement and hardware complexity.

**REFERENCES:**

[1] Burton H. Bloom, "Space/Time trade-offs in Hash coding with Allowable Errors, *Communication of the ACM*, Vol.13, No.7, 1970, pp. 422-426.

[2] A.Broder and M.Mitzenmacher, "Network Application of Bloom Filter: A Survey", *Internet Mathematics*, Vol.1, No.4, 2004, pp. 484-509.

[3] Shaha beddin Geravand and Mahmood Ahmadi, "Bloom filter applications in network security: A state-of-the-art survey", *Computer Networks*, Vol.57, No.18, 2013, pp. 4047-4064.

[4] Christian Esteve Rothenberg, Carlos Alberto Braz Macapuna, Maurício Ferreira Magalhaes, Fabio Luciano Verdi and Alexander Wiesmaier, "In-packet Bloom filters: Design and networking applications", *Computer Networks*, Vol.55, 2011, pp. 1364–1378.

[5] A.Broder and M.Mitzenmacher, "Using Multiple Hash Function to Improve IP Lookups", *in proceedings of IEEE INFOCOM*, April 22-26, 2001, pp. 1454-1463.

[6] Gianni Antichi and Domenico, "Counting Bloom Filter for pattern matching and Anti-Evasion at the wire speed", *IEEE Network,* Vol.23, No.1, 2009, pp. 30-35.

[7] Deke Guo, Yunhao Liu, Xiang Yang Li, and Panlong Yang, "False Negative Problem of Counting Bloom Filter", *IEEE Transactions on Knowledge and Data Engineering*, Vol.22, No.5, 2010, pp. 651 – 664.

[8] Elham Safi, Andreas Moshovos and Andreas Veneris, "L-CBF: A Low-Power, Fast Counting Bloom Filter Architecture", *IEEE Transactions on Very Large Scale Integration (VLSI) systems*, Vol.16, No.6, 2008, pp. 628-638.

[9] Zhang Jin, Wu Jiang Xing and Lan Julong Liu Jianqiang, "Performance Evaluation and Comparison of Three Counting Bloom Filters, Journal of Electronics, Using Multiple Hash Function to Improve IP Lookups", *in proceedings of IEEE INFOCOM,* Vol. 26, No.3, 2009,pp. 332-340.

[10] Greg Martin and Kevin O'Bryant, "Constructions of Generalized Sidon Sets", *Journal of Combinatorial Theory, Series A,* Vol. 113, 2006, pp. 591–607.

[11] DW. Clark, "Maximal and Near-Maximal Shift Register Sequences: Efficient Event Counters and Easy Discrete Logarithms", *IEEE T Comput.* Vol.43, No.5, 2002, pp. 560-568.

Journal of Theoretical and Applied Information Technology
20th August 2014. Vol. 66 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

*Table 1: Weighted Sum Of $B_h$ Sequence*

| B2 sequence | Output | B3 sequence | Output | B4 sequence | Output |
|---|---|---|---|---|---|
| 1+1 | 2 | 1+1+1 | 3 | 1+1+1+1 | 4 |
| 1+4 | 5 | 1+1+4 | 6 | 1+1+1+4 | 7 |
| 1+11 | 12 | 1+1+11 | 13 | 1+1+1+11 | 14 |
| 1+13 | 14 | 1+1+13 | 15 | 1+1+1+13 | 16 |
| 4+4 | 8 | 1+4+4 | 9 | 1+1+4+4 | 10 |
| 4+11 | 15 | 1+4+11 | 16 | 1+1+4+11 | 17 |
| 4+13 | 17 | 1+4+13 | 18 | 1+1+4+13 | 19 |
| 11+11 | 22 | 1+11+11 | 23 | 1+1+11+11 | 24 |
| 11+13 | 24 | 1+11+13 | 25 | 1+1+11+13 | 26 |
| 13+13 | 26 | 1+13+13 | 27 | 1+1+13+13 | 28 |
| | | 4+4+4 | 12 | 1+4+11+11 | 27 |
| | | 4+4+11 | 29 | 1+4+11+13 | 29 |
| | | 4+4+13 | 21 | 1+4+13+13 | 31 |
| | | 4+11+11 | 26 | 4+4+4+4 | 16 |
| | | 4+11+13 | 28 | 4+4+4+11 | 23 |
| | | 4+13+13 | 30 | 4+4+4+13 | 25 |
| | | 11+11+11 | 33 | 4+4+11+11 | 30 |
| | | 11+11+13 | 35 | 4+4+11+13 | 32 |
| | | 11+13+13 | 37 | 4+4+13+13 | 34 |
| | | 13+13+13 | 39 | 4+11+11+11 | 37 |
| | | | | 4+11+11+13 | 39 |
| | | | | 4+11+13+13 | 41 |
| | | | | 4+13+13+13 | 43 |
| | | | | 11+11+11+11 | 44 |
| | | | | ..... | |

*Table 3: Comparison Of Throughput, Total Number Of Entries And Luts Consumed For $B_h$ Sequence Implementation For K=2 And K=5 Hashing*

| Method | Power in µw | | Throughput (Gbps) | | Number of entries | LUTs | |
|---|---|---|---|---|---|---|---|
| | k=2 | k=5 | k=2 | k=5 | | k=2 | k=5 |
| $B_h$ architecture (8 bits/cycle) | 1.409 | 1.785 | 3.26 | 2.81 | 128 | 331 | 428 |
| | 4.549 | 5.112 | 3.02 | 2.61 | 256 | 567 | 701 |
| | 6.708 | 6.523 | 2.72 | 2.47 | 512 | 890 | 915 |
| | 10.980 | 15.977 | 2.01 | 1.89 | 1k | 1219 | 1990 |
| CBF architecture (8 bits/cycle) | 1.112 | 1.385 | 2.6 | 2.18 | 128 | 215 | 276 |
| | 3.549 | 4.132 | 2.4 | 2.01 | 256 | 420 | 563 |
| | 5.718 | 6.115 | 1.87 | 1.47 | 512 | 710 | 854 |
| | 9.910 | 12.916 | 1.8 | 1.09 | 1k | 998 | 1311 |